Pràctiques integrades

Pràctica 8

Matemàtiques, curs 2001-2002.

8 Programació: lògica, iteracions i procediments

Alguns cops el Maple no tindrà les comandes que necessitem per a algun treball concret. En aquests casos necessitarem definir les nostres pròpies comandes i necessitarem saber com incloure funcions lògiques, iteracions i procediments. Comencem amb funcions lògiques:

8.1 Lògica: `if-then-else-end if`

La comanda bàsica és `if` i podem veure el funcionament al següent exemple:

Exemple 8.1

Suposem que volem definir una funció `f(x)` que valgui -1 si $x < 0$, 0 si $x = 0$ i 1 si $x > 0$. Podríem fer-ho de la següent manera:

```
> restart;
> f:=x-> if x<0 then -1 elif x=0 then 0 else 1 end if;
```

comproveu que funciona executant:

```
> f(-.5);f(0);f(.5);
```

Nota: el que hem definit ja estava definit al Maple amb el nom de `signum( )`.
El que hem fet és definir la funció `f` on hauríem de traduir `if` per "*si*", `elif` per "*altrament si*" i `end if` per "*hem acabat*". Llavors podem pensar la comanda `if` com una comanda amb tres elements:

- El primer que hem de veure és la sintaxi de la funció. Hem d'entrar els arguments `if ... then`, `elif ... then`, `else`, i `end if` sempre i en aquest ordre.
- El segon és la introducció de condicions lògiques. Això són expressions com $x > 0$, ...i que en Maple s'han d'introduir com:

`<`	més petit que,
`<=`	més petit o igual que,
`>`	més gran que,
`>=`	més gran o igual que,
`=`	igual,
`<>`	diferent.

Quan Maple es troba amb alguna d'aquestes expressions ho avalua com una "*expressió Booleana*", o sigui, com una expressió que pot ser "*certa*" o "*falsa*". Això també es pot avaluar mitjançant la funció `evalb( )`:

```
> evalb(7>5);evalb(3=4);evalb(3<>4);
```

Podem fer construccions més complicades, combinant amb les comandes lògiques `and`, `or`, `xor` i `not`. Podem consultar la ajuda per a més detalls: `?boolean`.

- El tercer element són les comandes que s'han d'executar a cada un dels casos. En l'exemple tant sols havíem d'introduir -1, 0 i 1, però es poden introduir expressions més complexes: volem definir una funció `g` que prengui els següents valors:

  ```
  > g:=x-> if x<0 then 1/(1+x^2) else cos(x) end if;
  ```

  Podem comprovar que funciona

  ```
  > g(-1.5);g(2.5);
  ```

  Tot i això, què passa quan fem:

  ```
  > g(Pi);
  ```

  El missatge d'error diu que no ha pogut avaluar la condició booleana que hem introduït. Ens a passat degut a que `Pi` no és exactament un número. Obtenim exactament el mateix error quan avaluem `g` a una indeterminada a:

  ```
  > g(a);
  ```

  Una manera de solventar el problema seria utilitzant la comanda `evalf()` i avaluar `g` en un valor aproximat de `Pi`.

  ```
  > g(evalf(Pi));
  ```

  Intentem ara dibuixar la funció `g` que hem definit:

  ```
  > plot(g(x),x=-10..10);
  ```

  Tornem a obtenir errors en les expressions booleanes. El problema és que quan substitueix x a la comanda `g` encara no és un número i per això retorna l'error. En aquest cas el que podem fer és introduir la "*notació d'operadors*", o sigui, no escriure la variable `x`:

  ```
  > plot(g,-10..10);
  ```

  Però si volem utilitzar la funció `g` per a definir altres funcions, per exemple, si volem calcular la seva derivada, també tindrem problemes:

  ```
  > diff(g(x),x);
  ```

La funció `if` és molt útil en segons quins contextes, però no és la més apropiada per a definir funcions a trossos. Per a fer això és més versàtil la funció `piecewise( )`. Podem redefinir la funció `g` com:

```
> g:=x->piecewise(x<0, 1/(1+x^2),x>=0, cos(x));
```

Ara podem treballar amb `g` com amb qualsevol altre funció, per exemple:

```
> plot(g(x),x=-5..5);
```

i calcular la seva derivada:

```
> diff(g(x),x);
```

Per tant per a definir funcions a trossos utilitzarem la funció `piecewise`, mentre que reservarem `if-then-elif-end if` per a utilitzar-la dins de iteracions, ...

Exercici 8.1

Utilitzeu la funció `piecewise` per a definir amb Maple la funció que avalua la següent funció i feu la gràfica a l'interval $(-4, 4)$.

$$f(t) = \begin{cases} \frac{(2+t)^3}{6} & \text{si } t \in [-2,-1) \\ \frac{-3t^3-6t^2+4}{6} & \text{si } t \in [-1,0) \\ \frac{3t^3-6t^2+4}{6} & \text{si } t \in [0,1) \\ \frac{-(t-2)^3}{6} & \text{si } t \in [1,2) \\ 0 & \text{si } t \notin [-2,2) \end{cases}$$

Segons com feu la definició haureu d'introduir rangs amb límit inferior i límit superior. Per a això hem d'introduir dues desigualtats, i això ho hem de fer mitjançant la comanda `and`.

8.2 Iteracions: `for`, `do`, `end do` i `while`

Moltes comandes a Maple contenen iteracions: per exemple, les comandes `sum`, `fsolve`, ... Tot i això algun cop necessitarem definir les nostres pròpies iteracions. Com a exemple, calculem la suma dels enters entre 1 i 100 elevats al quadrat:

```
> sum(i^2,i=1..100);
```

338350

En alguna part del procés Maple ha aplicat una iteració per a fer el càlcul. Fet "a mà" hauríem fet:

Iteració suma:

Decidim primer el nom de la variable, i la inicialitzem a zero.

```
> resp:=0;
```

Llavors cal introduir la iteració. Per a això utilitzem les comandes `for..from..by..to..while..do..end do`. Les comandes que s'han d'executar s'han d'introduir entre `do` i `end do`.

```
> for i from 1 to 100 do
  resp:=resp + i^2;
  end do;
```

Per a evitar la sortida de tots els passos, podem canviar el punt i coma (`;`) per uns dos punts (`:`) i demanar el valor de la variable `resp` al final del procediment.

```
> resp:=0;
```

```
> for i from 1 to 100 do
  resp:=resp + i^2;
  end do:
```

```
> resp;
```

Exercici 8.2

Calculeu la suma de tots els nombre senars entre 1 i 99 mitjançant la comanda `for` (consulteu l'ajuda per veure la opció `by` dins de la comanda `for`).

Més iteracions:

A vegades no coneixerem el nombre d'iteracions que volem fer. La informació que tindrem serà que hem d'iterar fins que alguna condició fixada es doni. Per exemple, què passa si apretem indefinidament la tecla "*cosinus*" de la vostra calculadora, començant pel valor 5? A l'exemple ho farem 20 vegades:

```
> x:=5.;

> for i from 1 to 20 do
  x:=cos(x);
  end do;
```

Podeu observar que sembla que la successió es va acostant a un valor determinat. De fet, podeu comprovar que aquest valor ha de complir l'equació $\cos(x) = x$. Tot i això, podeu veure que amb 20 cops no n'hi ha prou per a obtenir una successió que no varii (8 dígits correctes), per tant, el que volem fer és iterar el procediment fins que la diferència entre dos passos sigui inferior a un número fixat, per exemple, 10^{-8}.

Necessitem dues variables, una serà `x` i la segona serà `xold`, que en tot moment serà el valor de la iteració anterior (haurem de donar-li un valor inicial diferent del que donem a `x`).

```
> x:=5.;xold:=0;
```

Llavors hem d'iterar mentre es compleixi que $|\texttt{x} - \texttt{xold}| > 10^{-8}$. La comanda la podríem entrar com:

```
> for i from 1 while abs(x-xold)>1e-8 do
  tempx:=cos(x):
  xold:=x:
  x:=tempx:
  end do;
```

Observem que la "`i`" que hi ha dins del `for` no s'ha utilitzat al procediment. De fet en qualsevol iteració l'única part obligatòria és el `do..end do`. Les parts `for` i `while` són opcionals. Com a exemple podríem escriure:

```
> x:=0.; xold:=1;
  while abs(x-xold)>1e-8 do
  tempx:=cos(x):
  xold:=x:
  x:=tempx:
  end do;
```

8.3 Procediments

Quan volem definir una funció poder necessitarem incloure iteracions, condicions lògiques, altres funcions ... in en general potser necessitarem dues o més instruccions de Maple encadenades per a fer els càlculs. També és bastant comú que ens aquests casos utilitzem variables que només tenen sentit dins el càlcul de la funció que estem definint i no a la resta del full de treball. Això és el que anomenarem **variables locals**, mentre que les que són vàlides a tot el full de treball les anomenarem **variables globals**.
Per a definir funcions que incluen variables locals i globals hem d'utilitzar la funció `proc`.

Exemple 8.2

Mireu l'estructura de la següent definició:

```
> f:=proc(x,y)
  local a,b,z;
  global d;
  a:=3.;b:=17.;
  z:=a*b*x*y*d;
  end proc;
```

Podem veure com s'executa aquesta definició provant:

```
> f(5,5);
> d:=2.;
> f(5,5);
> d:='d';
> f(5,5);
> a;
> b;
> z;
```

Exemple 8.3

La funció següent avalua el cosinus d'un angle en graus:

```
> cosdeg:=proc(theta)
  local resultat,phi;
  phi:=theta*Pi/180;
  resultat:=cos(phi);
  end proc;
```

Proveu calculant el cosinus de 45 graus.
Si voleu veure els passos que segueix un procediment, ho podeu fer amb la funció `trace`. Aquesta funció permet "*activar*" per pantalla els passos que hi ha dins de la funció `proc`.

Exemple 8.4

Per a activar la funció `cosdeg` hem de fer:

```
> trace(cosdeg);
```

llavors l'execució fa:

```
> cosdeg(45);
```

Si volem desactivar aquesta la visualització podem fer:

```
> untrace(cosdeg);
```

8.4 Exercicis

Els següents exercicis són perquè practiqueu els diferents conceptes de "*programació*" que hem introduït, però també pretenen il·lustrar alguns conceptes matemàtics. No us oblideu d'interpretar els resultats que aneu obtenint.

Exercici 8.3

Considereu la funció:

$$\texttt{aprcos}(x,n) = \sum_{i=0}^{n}(-1)^i \frac{x^{2i}}{(2i)!}$$

on x és un paràmetre real i n és un enter positiu.
Dibuixeu en on sol gràfic les funcions `cos(x)`, `aprcos(x,1)`, `aprcos(x,2)` i `aprcos(x,4)` per a `x` entre -2π i 2π.

Exercici 8.4

Definiu una funció que depengui de tres paràmetres `aprox(funcio,x,iteracions)` que retorni una llista amb els valors `[x+1/i,f(x+1/i)]` per a `i=1..iteracions`.
Avalueu `aprox` a la funció $\frac{\sin(x)}{x}$, per a $x = 0$ amb 50 iteracions.
Modifiqueu la funció `aprox` per a que a més retorni un dibuix dels punts calculats.

Exercici 8.5

Definiu una funció `ordre` que, donada una permutació σ, calculi el seu ordre.
Calculeu l'ordre de les permutacions $\sigma_1 = (1,3,5)(2,4)$ i $\sigma_2 = (1,2,3,4,5)(6,7,8)$ utilitzant la funció `ordre` que acabeu de definir.